STAT 672 Final Project: Stochastic Gradient Descent

Tom Wallace

May 3, 2018

Contents

Chapter 1

Introduction

1.1 Organization

This paper is divided into four sections. The remainder of this **Introduction** section gives intuitive motivation for stochastic gradient descent (SGD). The **Method and Theory** section more rigorously derives SGD, outlines convergence properties, and reviews extensions to the basic algorithm. The **Applications** sections highlights SGD's role in statistical learning, including a simulation analysis to demonstrate SGD's utility in large-n settings. The **Conclusion** section summarizes overall findings.

1.2 Motivation

SGD is an optimization algorithm commonly used for fitting coefficients in statistical models, particularly in non-parametric and large-n settings. Before covering exactly how SGD works, we contextualize its advantages in these areas.

1.2.1 Parametric vs. non-parametric approaches

Consider a feature matrix $\boldsymbol{X}$ and outcome variable $\boldsymbol{Y}$, and the task of fitting coefficients in a model to predict Y_i based on $\boldsymbol{X}_i$. Assuming that particular model elements follow a known statistical distribution aids the estimation of coefficients. For example, assumptions in ordinary least squares (OLS) regression—assumptions that readers almost certainly are familiar with and so will not be repeated here—allow a closed form solution, $\hat{\boldsymbol{\beta}} = (\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{Y}$.

Even if a parametric model does not have a closed-form solution, the parametric assumption allows some useful optimization techniques. Consider logistic regression. The maximum likelihood estimator (MLE) approach for estimating coefficients leads to a system of D equations. This system of equations typically is numerically solved using the iterative Newton-Raphson algorithm:

$$\hat{\boldsymbol{\beta}}^{(t+1)} = \hat{\boldsymbol{\beta}}^{(t)} - \boldsymbol{H}^{-1}(\hat{\boldsymbol{\beta}}^{(t)})\boldsymbol{J}(\hat{\boldsymbol{\beta}}^{(t)})$$

(t) refers to a particular iteration, $\boldsymbol{J}$ is the Jacobian of the log likelihood function l, and $\boldsymbol{H}$ is the Hessian of l. The practicality of Newton-Raphson thus depends on whether it is convenient to find $\boldsymbol{J}$ and in particular $\boldsymbol{H}$. It is convenient for logistic regression

because parametric and independent-and-identically-distributed (IID) assumptions mean l is a simple sum of the log probability distribution function (PDF) for each observation. We "know" (assume) the form of this PDF and so are confident that the second derivative exists and is not onerous. In non-parametric settings, we often cannot be so certain and face the possibility of $\boldsymbol{H}$ being non-existent or cumbersome.

The need to conduct optimization in non-parametric settings is a chief motivation for gradient descent (GD), of which SGD is a variant. SGD does not require any parametric assumptions. In its most basic form, SGD only requires finding the gradient (though some extensions do need the Hessian or an approximation to it). SGD thus is well-suited for supervised and unsupervised statistical learning, where parametric assumptions often are not made and Newton-Raphson and other second-order methods often are undesirable.

1.2.2 Computational complexity

How an optimization technique scales with sample size n is another important consideration. It is little comfort if a method reaches the correct solution but requires an excessive amount of time to do so. "Plain" or "batch" GD requires evaluating the gradient for every observation, every iteration, until the algorithm converges. For example, for a dataset of $n = 10^6$ that required 25 iterations to converge, batch GD would require evaluating the gradient 25×10^6 times. This scaling with n can cause untenably long computation time. Similarly, note that if the Jacobian of some function has D elements, the Hessian will have D^2. In high-dimensional settings, this too can be computationally intractable.

In contrast, SGD evaluates the gradient for only a single randomly chosen observation per iteration. This approach means convergence is "noisier" and hence requires more iterations to converge, but each iteration is less compute-intensive and so can be done faster. SGD thus scales more favorably with n than does GD and so is useful for large-n applications.

Chapter 2

Method and theory

2.1 Basic form

Suppose we want to minimize a function $f : \mathbb{R}^D \to \mathbb{R}$.

$$\min_w f(w) \tag{2.1}$$

f takes as input $w \in \mathbb{R}^D$. Assume that f is convex and differentiable, i.e. we can compute its gradient with respect to w, $\nabla f(w)$. The optimal w, i.e. that which minimizes (2.1), is denoted $\tilde{w}$. We are willing to settle for a close approximation $\hat{w}$. The iterative GD algorithm for finding $\hat{w}$ is:

$$w^{(t+1)} = w^{(t)} - \gamma \nabla f(w^{(t)}) \tag{2.2}$$

t refers to a particular iteration. Assume that we have supplied an initial starting guess $w^{(0)}$. γ refers to step size (also called learning rate). Assume for now that γ is fixed. The GD algorithm iterates until some stop condition is met. This may be a fixed number of iterations, or that the quality of approximation meets some predefined threshold. A common stopping condition is when the L2 norm of the gradient is less than some arbitrarily small constant. Obviously, the smaller the constant, the closer $\hat{w}$ will be to $\tilde{w}$.

$$||\nabla f(w^{(t)})||_2 \leq \epsilon \tag{2.3}$$

Consider a modified situation. Suppose f now takes two arguments, w and $X \in \mathbb{R}^D$. $f : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$. We have n observations of X, and denote by $\boldsymbol{X}_{n \times D}$ the matrix of these observations, with X_i being a row in that matrix. We apply f over all observations, i.e. $\frac{1}{n} \sum_{i=1}^{n} f(w, X_i)$. Our new problem is:

$$\min_w \frac{1}{n} \sum_{i=1}^{n} f(w, X_i) \tag{2.4}$$

We could apply the GD algorithm above to find $\hat{w}$.

$$w^{(t+1)} = w^{(t)} - \gamma \frac{1}{n} \sum_{i=1}^{n} \nabla f(w^{(t)}, X_i) \tag{2.5}$$

But, note that doing so requires evaluating the gradient at every single observation $i \leq n$. This may be computationally intractable for large n and high-dimensional D. The innovation of SGD is to instead evaluate only a single randomly-chosen i at each iteration t.

$$w^{(t+1)} = w^{(t)} - \gamma \nabla f(w^{(t)}, X_i) \tag{2.6}$$

As it turns out, SGD asymptotically converges to $\tilde{w}$ and does so in a computationally advantageous way compared to GD.

2.2 Key properties

2.2.1 Correctness

This section gives intuition and partial proof for why SGD converges to $\tilde{w}$.

Intuitively, our assumption that f is convex means that there is one critical point and it is the global minimum. Basic calculus tells us that this critical point is located where $f' = 0$ (in one dimension) or $\nabla f = 0$ (in higher dimensions). So our task is to search in the space of f for that point. We start with some initial guess $w^{(0)}$, and every iteration, move "down" the gradient in search of zero. Every iteration, we check if the gradient is equal to 0; if not, we keep moving "down." If the gradient is arbitrarily close to zero, then we have found the critical point, which must be the value of w that minimizes f and hence is $\tilde{w}$ (or at least a close approximation to it).

For a more formal proof—following that of Tibshirani 2013—assume (in addition to previously stated assumptions that f is convex and differentiable) that the gradient of f is Lipschitz-continuous with constant L, i.e. that $||\nabla f(x) - \nabla f(y)||_2 \leq L||x - y||_2$ for any x, y. This implies that $\nabla^2 f(w) - LI$ is a negative semi-definite matrix. We perform a quadratic expansion of f around $f(x)$ and obtain the following inequality:

$$\begin{aligned} f(y) &\leq f(x) + \nabla f(x)^T (y - x) + \frac{1}{2} \nabla^2 f(x) ||y - x||_2^2 \\ &\leq f(x) + \nabla f(x)^T (y - x) + \frac{1}{2} L ||y - x||_2^2 \end{aligned} \tag{2.7}$$

Now, suppose we use the GD algorithm presented in (2.2) with $\gamma \leq \frac{1}{L}$. Denote $x^+ = x - \gamma \nabla f(x)$ and substitute x^+ in for y:

$$\begin{aligned} f(x^+) &\leq f(x) + \nabla f(x)^T (x^+ - x) + \frac{1}{2} L ||x^+ - x||_2^2 \\ &= f(x) + \nabla f(x)^T (x - \gamma \nabla f(x) - x) + \frac{1}{2} L ||x - \gamma \nabla f(x) - x||_2^2 \\ &= f(x) - \nabla f(x)^T \gamma \nabla f(x) + \frac{1}{2} L ||\gamma \nabla f(x)||_2^2 \\ &= f(x) - \gamma ||\nabla f(x)||_2^2 + \frac{1}{2} L \gamma^2 ||\nabla f(x)||_2^2 \\ &= f(x) - (1 - \frac{1}{2} L \gamma) \gamma ||\nabla f(x)||_2^2 \end{aligned} \tag{2.8}$$

We have defined $\gamma \leq \frac{1}{L}$. This implies:

$$-(1 - \frac{1}{2}L\gamma) = \frac{1}{2}L\gamma - 1 \leq \frac{1}{2}L\frac{1}{L} - 1 = \frac{1}{2} - 1 = -\frac{1}{2}$$

Returning to (2.8), we obtain:

$$f(x^+) \leq f(x) - \frac{1}{2}\gamma||\nabla f(x)||_2^2 \tag{2.9}$$

A squared L2 norm will always be positive unless its content is 0, and we have defined γ to be positive, so $\frac{1}{2}\gamma||\nabla f(x)||_2^2$ will always be positive unless the gradient is equal to zero. So, (2.9) implies that GD results in a strictly decreasing objective function value until it reaches the point where the gradient equals zero, which is the optimal value. A key caveat is an appropriately chosen γ, a point covered in more detail later.

The above proof is for GD. Proof is not given for why SGD also converges to the optimal value. Informally, we can note that the above proof says given infinite t and appropriate γ, the iterative algorithm will always converge to the optimal value. SGD is doing the same thing as GD, just with a single observation per iteration rather than the entire dataset per iteration. Since SGD is using less information per iteration, we would expect the convergence to require more iterations. But per (2.9) we expect it to *eventually* arrive at the optimal solution. If so, the difference between GD and SGD must then primarily revolve around convergence speed, not the final value that is converged to.

2.2.2 Speed

Table 2.1—a simplified version of that in Bottou 2010—illustrates the computational advantages of SGD over GD. As shown in row 1, because SGD uses less information per iteration than GD, it requires more iterations to achieve fixed degree of accuracy ρ. However, row 2 shows that because GD computes the gradient for every observation every iteration, while SGD only does so for a single randomly observation, GD's time per iteration scales linearly with n while SGD's is a constant. Thus, we conclude that SGD's time to reach accuracy ρ scales only with ρ, while GD's scales with both ρ and—crucially—linearly with n. So, the larger n grows, the larger SGD's computational advantage over GD.

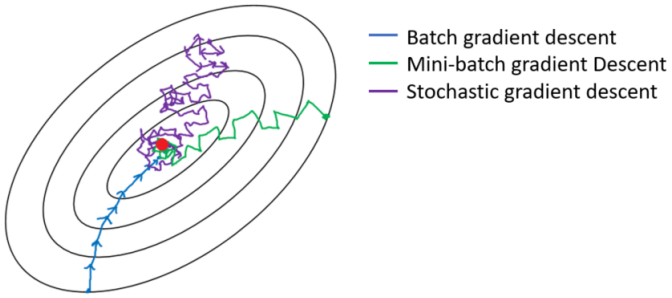
Table 2.1: Asymptotic comparison of GD and SGD

	GD	**SGD**
Iterations to accuracy ρ	$\log \frac{1}{\rho}$	$\frac{1}{\rho}$
Time per iteration	n	1
Time to accuracy ρ	$n \log \frac{1}{p}$	$\frac{1}{\rho}$

Figure 2.1 visually compares SGD's convergence to that of GD (and mini-batch gradient descent, a variation addressed later in this paper).[1] The path of SGD's convergence is very squiggly, reflecting the extra variation introduced by only using a single observation per iteration. The path of GD's convergence is much smoother, reflecting the use of all observations per iteration.

[1]Image taken from www.towardsdatascience.com

Figure 2.1: Convergence



2.3 Extensions

The basic SGD algorithm has been extended in many different ways. The popularity of the algorithm disallows a comprehensive treatment of all developments. This sub-section covers some of the more important ones.

2.3.1 Step size

Much depends on an appropriately chosen step size γ. If γ is too small, gradient descent may take an excessively long time to run (since we are only moving a small distance "down" the gradient every iteration); if γ is too large, then an iteration's step may "overshoot" the critical point. An entire literature has developed around the best way to select γ. The central insight that is common to most γ-selection methods is to treat it as a time-indexed rather than fixed hyperparameter, i.e., $\gamma^{(t)}$ rather than γ. Ideally we would like $\gamma^{(t)}$ to be large when far away from the critical point (so as to increase speed of convergence) and to be small when close to the critical point (so as to avoid overshooting).

Line search is one method for computing step size. As described in Boyd and Vandenberghe 2004, there are two main variants: *exact* and *backtracking*. In exact line search, $\gamma^{(t)}$ is chosen to minimize f along the ray $\{x+\gamma^{(t)}\Delta x\}$, where $\gamma^{(t)} \in \mathbb{R}^+$) and Δx is the descent direction determined by SGD:

$$\gamma^{(t)} = \underset{s \geq 0}{\operatorname{argmin}}\, f(x + s\Delta x) \tag{2.10}$$

This is an optimization problem in and of itself, and so it can be computationally impractical to add this burden in addition to the "main" problem we are trying to solve using SGD. *Backtracking* is an iterative approximation method that is computationally lighter. The backtracking algorithm takes two hyper-parameters $\alpha \in (0, 0.5)$ and $\beta \in (0, 1)$. It starts with an initial guess of $s^{(0)} = 1$, and then for every iteration t, updates $s := \beta t$. The algorithm stops when $f(x + s\Delta x) \leq f(x) + \alpha s \Delta f(x)^T \Delta x$, and the final value of s is taken as $\gamma^{(t)}$.

Bottou 2012 advocates using learning rates of the form $\gamma^{(t)} = \gamma^{(0)}(1 + \gamma^{(0)}\lambda t)^{-1}$. When the Hessian H of f is strictly positive, using $\gamma^{(t)} = (\lambda_{\min} t)^{-1}$, where $\lambda_{\min}$ is the smallest eigenvalue of H, produces the best convergence speed. Note that $(\lambda_{\min} t)^{-1}$ decreases asymptotically with t, matching our intuition about larger step sizes in early iterations and

smaller step sizes in later iterations. However, simply using $\gamma^{(t)} = (\lambda_{\min} t)^{-1}$ can produce *too* large of steps in early iterations. Hence, it often works better to start with some reasonable initial estimate $\gamma^{(0)}$ that then decays in the fashion of $(\lambda_{\min} t)^{-1}$. By definition, "reasonable" is more of a judgment call than a mathematical statement: Bottou provides some heuristics for creating such an estimate of $\gamma^{(0)}$.

2.3.2 Momentum and acceleration

In our set-up of SGD, we stipulated that the objective function is convex. However, SGD is widely used in applications—most prominently, neural networks and deep learning—where this assumption is not always true. In such settings, it is possible that there are multiple local minima, and/or that there are areas where the surface curves much more steeply in one dimension than another. The challenge for SGD is how to avoid getting "stuck" in these local ravines and continue iterating until the true global minimum is achieved. Even if there is no pathological curvature, incorporating information about the local topology may result in more efficient algorithms and faster convergence.

Momentum—also called acceleration—is a common modification of GD. As outlined in Rumelhart, Hinton, McClelland, et al. 1986 and Qian 1999, we add a new acceleration term to the familiar GD equation. Let $z^{(t+1)} = \alpha z^{(t)} + \nabla f(w^{(t)})$. GD with momentum then is:

$$w^{(t+1)} = w^{(t)} - \gamma z^{(t+1)} \tag{2.11}$$

Now, the weight vector obtained for the current timestep depends both on the current gradient *and the update of the previous time-step*, with the acceleration parameter α governing how much weight is given to the previous time-step. This formulation leads to momentum: dimensions whose gradients point in the same direction have proportionally greater updates, while dimensions whose gradients exhibit strong curvature have proportionally lesser updates. The effect is to help stop SGD from oscillating in ravines and speed up convergence. There are many variations on acceleration, including Nesterov accelerated gradient (NAG) (Nesterov 1983), Adam (Kingma and Ba 2014), AdaGrad (Duchi, Hazan, and Singer 2011), and AdaDelta (Zeiler 2012).

2.3.3 Averaging

Averaged SGD (ASGD)—proposed in Polyak and Juditsky 1992—is another variation. Typically, we consider $\hat{w}$ to be the final iteration of SGD, when some stop condition is met. For example, in a situation where we run N iterations, $\hat{w} = w^{(N)}$. In ASGD, we consider $\hat{w}$ the average across all iterations.

$$\hat{w} = \bar{w} = \frac{1}{N} \sum_{t=1}^{N} w^{(t)} \tag{2.12}$$

The value proposition of ASGD is that sometimes SGD will oscillate around the optimal solution. By taking the average across all updates, ASGD reduces this noise and is likely to give a solution closer to the optimum. More sophisticated versions of ASGD have been proposed in Zhang 2004 and Xu 2011.

2.3.4 Mini-batch

Mini-batch gradient descent (MGD) is a mid-point between batch and stochastic gradient descent: it uses more than one but less than all observations per iteration. Suppose we have n observations and divide them into m groups called mini-batches, each of equal size $b = \frac{n}{m}$. Now, at every iteration t, randomly pick one of the mini-batches (all are equally likely to be chosen). For that mini-batch:

$$w^{(t+1)} = w^{(t)} - \gamma \frac{1}{b} \sum_{i=1}^{b} \nabla f(w^{(t)}) \tag{2.13}$$

MGD iterates with t and converges to $\hat{w}$ in the normal manner. Every update, a new b is randomly chosen and the above algorithm iterated. Dekel et al. 2012 and Li et al. 2014 analyze MGD's speed of convergence and suggest that has some advantageous qualities for parallelization, a topic that is explained in more depth in the following subsection.

2.3.5 Parallelization

SGD is common in large-n applications. Thus, even though SGD is a computational improvement over batch GD, there has been interest in whether SGD can be made even faster via parallelization. Zinkevich et al. 2010 present a technique for doing so. Though backed by very technical proofs, the actual algorithm is simple. Suppose we have N processors. We seek to use SGD to estimate w across n observations with fixed learning rate γ for fixed number of steps T.

Algorithm 1: Parallel SGD

Define $T = \frac{n}{N}$;
Randomly partition examples, giving T to each machine;
forall $i \in \{1 \dots N\}$ **do**
    Randomly shuffle data on machine i;
    Initialize $w^{i,(0)} = 0$;
    forall $t \in \{1 \dots T\}$ **do**
        $w^{i,(t+1)} = w^{i,(t)} - \gamma \nabla f(w^{i,(t)})$;
    end
end
Aggregate from all computers: $\bar{w} = \frac{1}{N} \sum_{i=1}^{N} w^{i,(T)}$;
return $\bar{w}$

Zinkevich et al. show that this algorithm has a number of desirable properties. It requires no communication between machines until the end; asymptotically, the error approaches zero; and, the amount of time required is independent of the number of examples. Further improvements have been proposed e.g., the Hogwild algorithm presented in Recht et al. 2011.

Chapter 3

Applications

In theory, SGD is topic-agnostic and so can be used in many different areas of optimization. In practice, SGD is overwhelmingly popular in the statistical learning field for estimating weight coefficients in predictive models.

3.1 SGD and statistical learning

Consider a typical supervised classification problem. We have feature matrix $\boldsymbol{X}_{n\times D}$ (n observations and D features) and labels $\boldsymbol{Y}_{n\times 1}, Y_i \in \{-1, 1\}$. The goal is to predict an observation's label Y_i using that observation's features $\boldsymbol{X}_i$. To emphasize the utility of SGD, suppose that both n and D are very large. We have a hypothesis class $\mathcal{F}$ of functions $f(\boldsymbol{w}, \boldsymbol{X}_i) \in \mathcal{F}$ parametrized by weight vector $\boldsymbol{w}$. We have convex loss function $L(Y_i, f(\boldsymbol{w}, \boldsymbol{X}_i))$ that expresses the cost of misclassification. We will consider the optimal function that which minimizes empirical risk over all observations: $\frac{1}{n}\sum_{i=1}^{n} L(Y_i, f(\boldsymbol{w}, \boldsymbol{X}_i))$. Denote $\hat{\boldsymbol{w}}$ the weight coefficients of this optimal function. We thus have:

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^{n} L(Y_i, f(\boldsymbol{w}, \boldsymbol{X}_i)) \tag{3.1}$$

How shall we solve this optimization problem? Newton's method requires calculating the Hessian, which will have D^2 entries; since D is large, this is inconvenient to compute. We could try batch gradient descent:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \gamma \frac{1}{n}\sum_{i=1}^{n} \nabla_{\boldsymbol{w}} L(Y_i, f(\boldsymbol{w}^{(t)}, \boldsymbol{X}_i)) \tag{3.2}$$

But, in line with our analysis in section 2.2.2, batch GD requires evaluating the gradient over every observation $i \leq n$ for every iteration t. Since we have defined n to be large, this is computationally inconvenient. Rather, we use SGD to evaluate only a single randomly chosen observation per iteration.

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \nabla_{\boldsymbol{w}} L(Y_i, f(\boldsymbol{w}^{(t)}, \boldsymbol{X}_i)) \tag{3.3}$$

Per previous analyses we know that this approach will converge to an arbitrarily good answer (if certain assumptions are met); will converge to that arbitrary degree of precision

in less time than plain gradient descent; does not necessarily require computing the Hessian; and can be parallelized across many machines. These attractive qualities make SGD well-represented in almost all areas of machine learning. In the following subsection, we take a closer look at one particularly prominent application, support vector machines.

3.1.1 Support vector machines

Originally proposed by Boser, Guyon, and Vapnik 1992 and Cortes and Vapnik 1995, support vector machines (SVM) are among the best supervised learning algorithms. Here we set up the SVM optimization problem and then show how SGD can be used to solve it.

Suppose we have labels $\boldsymbol{Y}$ with $Y_i \in \{-1, 1\}$ and features $\boldsymbol{X}_{n\times D}$, corresponding to n observations and D features. We want to construct a decision hyperplane $\langle \boldsymbol{w}, \boldsymbol{X}\rangle + w_0 = 0$. $\boldsymbol{w}$ is a weight vector that we apply to $\boldsymbol{X}$. If for a particular observation i $\langle \boldsymbol{w}, \boldsymbol{X}_i\rangle + w_0 < 0$, we classify $Y_i = -1$. Similarly, if $\langle \boldsymbol{w}, \boldsymbol{X}_i\rangle + w_0 > 0$, we classify $Y_i = 1$. We introduce slack variables ξ_i to allow some violation of the margin, as well as a cost parameter C to control them. The primal soft-margin SVM problem thus is:

$$\begin{aligned} \min_{\boldsymbol{w}\in\mathbb{R}^D, w_0\in\mathbb{R}, \boldsymbol{\xi}\in\mathbb{R}^n} & \frac{1}{2}||\boldsymbol{w}||^2 + C\frac{1}{n}\sum_{i=1}^{n}\xi_i \\ s.t. \quad & Y_i(\langle \boldsymbol{w}, \boldsymbol{X}_i\rangle + w_0) \geq 1 - \xi_i \\ & \xi_i \geq 0 \end{aligned} \tag{3.4}$$

This problem can be recharacterized through the lens of empirical risk minimization. Suppose we obtain an optimal answer $\boldsymbol{w}^*, w_0^*, \boldsymbol{\xi}^*$. Then:

$$\xi_i^* = \max(0, 1 - Y_i(\langle \boldsymbol{w}, \boldsymbol{X}_i\rangle + w_0))$$

We now can recharacterize the problem as:

$$\min_{\boldsymbol{w}, w_0} C\frac{1}{n}\sum_{i=1}^{n}\left[\max(0, 1 - Y_i(\langle \boldsymbol{w}, \boldsymbol{X}_i\rangle + w_0)) + \frac{1}{2}||\boldsymbol{w}||^2\right]$$

This is equivalent to:

$$\min_{f\in\mathcal{F}} \frac{1}{n}\sum_{i=1}^{n}\left[L(Y_i, \boldsymbol{w}, \boldsymbol{X}_i) + \frac{\lambda}{2}||\boldsymbol{w}||^2\right] \tag{3.5}$$

Where $\mathcal{F}$ is the linear hypothesis class, L is the hinge loss

$$\begin{aligned} L_{\text{hinge}}(Y_i, \boldsymbol{w}, \boldsymbol{X}_i) &= \max(0, 1 - Y_i f(\boldsymbol{w}, \boldsymbol{X}_i)) \\ f(\boldsymbol{w}, \boldsymbol{X}_i) &= \boldsymbol{w}^T\boldsymbol{X}_i + w_0 \end{aligned}$$

and $\lambda = \frac{1}{2C}$. We use $\frac{\lambda}{2}$ for reasons of convenience. This problem could solved using the Lagrangian dual, but SGD is a popular method to solve the primal formulation given here. First note that the hinge loss function is not differentiable. We must use a sub-gradient:

$$\frac{\partial L}{\partial \boldsymbol{w}} = \begin{cases} -Y_i\boldsymbol{X}_i & \text{if } Y_i f(\boldsymbol{w}, \boldsymbol{X}_i) < 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

The basic batch GD approach

$$\begin{aligned}\boldsymbol{w}^{(t+1)} &= \boldsymbol{w}^{(t)} - \gamma \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{w}} \left[L(Y_i, \boldsymbol{w}, \boldsymbol{X}_i) + \frac{\lambda}{2} ||\boldsymbol{w}||^2 \right] \\ &= \boldsymbol{w}^{(t)} - \gamma \frac{1}{n} \sum_{i=1}^{n} \left[L \nabla_{\boldsymbol{w}} (Y_i, \boldsymbol{w}, \boldsymbol{X}_i) + \lambda \boldsymbol{w} \right]\end{aligned}$$

thus becomes

$$\boldsymbol{w}^{(t+1)} = \begin{cases} \boldsymbol{w}^{(t)} - \gamma \frac{1}{n} \sum_{i=1}^{n} \left[\lambda \boldsymbol{w}^{(t)} - Y_i \boldsymbol{X}_i \right] & \text{if } Y_i f(\boldsymbol{w}^{(t)}, \boldsymbol{X}_i) < 1 \\ \boldsymbol{w}^{(t)} - \gamma \frac{1}{n} \sum_{i=1}^{n} \lambda \boldsymbol{w}^{(t)} & \text{otherwise} \end{cases} \tag{3.7}$$

As previously discussed, batch GD is computationally expensive and so for large n we prefer SGD, selecting only a single random observation per iteration.

$$\boldsymbol{w}^{(t+1)} = \begin{cases} \boldsymbol{w}^{(t)} - \gamma \left(\lambda \boldsymbol{w}^{(t)} - Y_i \boldsymbol{X}_i \right) & \text{if } Y_i f(\boldsymbol{w}^{(t)}, \boldsymbol{X}_i) < 1 \\ \boldsymbol{w}^{(t)} - \gamma \lambda \boldsymbol{w}^{(t)} & \text{otherwise} \end{cases} \tag{3.8}$$

There have been further efforts to improve SGD for the specific application of SVM, most notably the Pegasos algorithm of Shalev-Shwartz et al. 2011.

3.1.2 Neural networks and back-propagation

Neural networks are another class of statistical learning models for which SGD plays a fundamental role in estimating weights, in particular through a process called back-propagation. Space constraints disallow a detailed examination along the lines of that conducted for SVM. This subsection aims only to note that SGD is widely used with neural networks, and that this usage provides the practical motivation behind many of the extensions to the SGD algorithm covered in Section 2.3. This is not to say that these techniques are *solely* intended for neural networks—indeed, many of them precede neural networks' current popularity—but their utility for neural networks is part of why the relevant papers have thousands of citations.

Neural networks often use non-convex loss functions that have multiple local minima in which SGD can get "lost" while searching for global minimum. Acceleration and momentum (2.3.2) thus are required to achieve effective training, particularly on "deep" networks with many hidden layers (Sutskever et al. 2013). Neural networks are often extremely complex and high-dimensional and hence require enormous training data (LeCun, Bengio, and Hinton 2015); hence, speeding up computation via parallelization (2.3.5) or GPU computing is common.

3.2 Simulation case study

Simulation provides an example of the computational advantages of SGD. Suppose we have one-dimensional feature matrix $\boldsymbol{X}$ with $X_i \sim N(0, 1)$ and outcome $\boldsymbol{Y}$ with the relationship $Y_i \sim (0.5X_i) + \epsilon$. There are 10 million observations. We seek to estimate coefficients in a regression $y = \beta_0 + \beta_1 X$ by least squares. We do so via two competing methods: batch GD and SGD. Both algorithms were run on the $n = 10^6$ dataset with the same step size γ and precision ϵ. As shown in Table 3.1, both algorithms converge to essentially the correct answer, but SGD does so in dramatically less time than GD.

Figure 3.1: Generating data

```
import numpy as np
X = np.random.randn(int(10e6))
Y = (0.5 * X) + np.randn(1)
```

Figure 3.2: Batch GD algorithm

```
  def GD(X, Y, gamma, epsilon, maxiter):
  n = Y.size
  w = 1 #initial guess
  for i in range(maxiter):
    Yhat = np.dot(X, w)
    update = w - gamma * (np.dot(X.T, Yhat - Y) / n)
    if (abs(w - update) < epsilon):
      break
    else:
      w = update
  return(w)
```

Figure 3.3: SGD algorithm

```
def SGD(X, Y, gamma, epsilon, maxiter):
  n = Y.size
  w = 1 #initial guess
  for i in range(maxiter):
    j = np.random.randint(0, n)
    Yhat = w * X[j]
    update = w - gamma * ((Yhat - Y[j])*X[j])
    if (abs(w - update) < epsilon):
      break
    else:
      w = update
  return(w)
```

Table 3.1: Runtime comparison

	$\hat{\beta}_1$	**Time (s)**
GD	0.499	19.851
SGD	0.507	1.832

3.3 Real-world applications

If a Silicon Valley press release uses phrases like "artificial intelligence", "machine learning", and "big data" to describe a new product or service, it very likely relies upon SGD in some way. Such is the extent of SGD's popularity. Below are some prominent examples of SGD's practical applications.

3.3.1 ImageNet

ImageNet is a dataset consisting of millions of labeled images. The ImageNet Large Scale Visual Recognition Competition (ILSVRC) is an annual contest in which competing teams train models for automatic image classification. The submission of Krizhevsky, Sutskever, and Hinton 2012 used convolutional neural nets to achieve dramatic reduction in error rates and is one of the most cited machine learning papers of all time. Per the aforementioned paper: "We trained our models using stochastic gradient descent with a batch size of 128 examples, momentum of 0.9, and weight decay of 0.00005."

3.3.2 AlphaGo

Google's AlphaGo software applies deep learning to the board game of Go and has received extensive attention for its ability to defeat even top human players (e.g., twice making the cover of *Nature* with Silver, Huang, et al. 2016 and Silver, Schrittwieser, et al. 2017). Google's slides from the 2016 International Conference on Machine Learning (ICML) indicate that SGD is used in every stage of the model: supervised learning of policy networks, reinforcement learning of policy networks, and reinforcement learning of value networks.

3.3.3 Netflix grand prize

The Netflix Grand Prize was a competition hosted by Netflix to create better movie recommendation models (Bennett, Lanning, et al. 2007). Most of the top entries used SGD for estimating model coefficients; for example, Koren 2009 states that "in order to learn the involved parameters... learning is done by a stochastic gradient descent algorithm running for 30 iterations."

Chapter 4

Conclusion

SGD is an iterative optimization algorithm. SGD does not require parametric assumptions and is computationally attractive. These factors make it popular in statistical learning, where parametric assumptions often are not made and n and D often are large. There have been many extensions to the basic SGD algorithm and it remains a topic of active research. Outside of academia, SGD plays a central role in many real-world applications of statistical learning such as image recognition, artifical intelligence, and online personalization.

Chapter 5

Bibliography

Bennett, James, Stan Lanning, et al. (2007). "The netflix prize". In: *Proceedings of KDD cup and workshop*. Vol. 2007. New York, NY, USA, p. 35.

Boser, Bernhard E, Isabelle M Guyon, and Vladimir Vapnik (1992). "A training algorithm for optimal margin classifiers". In: *Proceedings of the fifth annual workshop on computational learning theory*. ACM, pp. 144–152.

Bottou, Léon (2010). "Large-scale machine learning with stochastic gradient descent". In: *Proceedings of COMPSTAT'2010*. Springer, pp. 177–186.

— (2012). "Stochastic gradient descent tricks". In: *Neural networks: Tricks of the trade*. Springer, pp. 421–436.

Boyd, Stephen and Lieven Vandenberghe (2004). *Convex optimization*. Cambridge university press.

Cortes, Corinna and Vladimir Vapnik (1995). "Support-vector networks". In: *Machine learning* 20.3, pp. 273–297.

Dekel, Ofer et al. (2012). "Optimal distributed online prediction using mini-batches". In: *Journal of Machine Learning Research* 13.Jan, pp. 165–202.

Duchi, John, Elad Hazan, and Yoram Singer (2011). "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of Machine Learning Research* 12.Jul, pp. 2121–2159.

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Koren, Yehuda (2009). "The bellkor solution to the netflix grand prize". In:

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*, pp. 1097–1105.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning". In: *Nature* 521.7553, p. 436.

Li, Mu et al. (2014). "Efficient mini-batch training for stochastic optimization". In: *Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, pp. 661–670.

Nesterov, Yurii (1983). "A method of solving a convex programming problem with convergence rate O(1/sqr(k))". In: *Soviet Mathematics Doklady* 27, pp. 372–376.

Polyak, Boris T and Anatoli B Juditsky (1992). "Acceleration of stochastic approximation by averaging". In: *SIAM Journal on Control and Optimization* 30.4, pp. 838–855.

Qian, Ning (1999). "On the momentum term in gradient descent learning algorithms". In: *Neural networks* 12.1, pp. 145–151.

Recht, Benjamin et al. (2011). "Hogwild: A lock-free approach to parallelizing stochastic gradient descent". In: *Advances in neural information processing systems*, pp. 693–701.

Rumelhart, David, Geoffrey Hinton, James McClelland, et al. (1986). "A general framework for parallel distributed processing". In: *Parallel distributed processing: Explorations in the microstructure of cognition* 1, pp. 45–76.

Shalev-Shwartz, Shai et al. (2011). "Pegasos: Primal estimated sub-gradient solver for svm". In: *Mathematical programming* 127.1, pp. 3–30.

Silver, David, Aja Huang, et al. (2016). "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587, pp. 484–489.

Silver, David, Julian Schrittwieser, et al. (2017). "Mastering the game of go without human knowledge". In: *Nature* 550.7676, p. 354.

Sutskever, Ilya et al. (2013). "On the importance of initialization and momentum in deep learning". In: *International conference on machine learning*, pp. 1139–1147.

Tibshirani, Ryan (Sept. 2013). *EECS 10-725: Optimization, Lecture 6*.

Xu, Wei (2011). "Towards optimal one pass large scale learning with averaged stochastic gradient descent". In: *arXiv preprint arXiv:1107.2490*.

Zeiler, Matthew D (2012). "ADADELTA: an adaptive learning rate method". In: *arXiv preprint arXiv:1212.5701*.

Zhang, Tong (2004). "Solving large scale linear prediction problems using stochastic gradient descent algorithms". In: *Proceedings of the twenty-first international conference on Machine learning*. ACM, p. 116.

Zinkevich, Martin et al. (2010). "Parallelized stochastic gradient descent". In: *Advances in neural information processing systems*, pp. 2595–2603.